

UFTP messages for flex trading at GOPACS

Last updated: 28-4-2025

GOPACS supports flex trading with Capacity Limiting Contracts (**CBCs**, *capaciteitsbeperkingscontracten*) and alternative non-firm transport rights such as **TDTR/NFA** through the exchange of Shapeshifter UFTP messages.

The implementation must be compliant to the [Shapeshifter UFTP specification](#) version **3.0.0** (for TDTR, **3.1.0** is required), with some restrictions which are described separately in this document.

Table of contents

- [CLC message flow](#)
- [TDTR/NFA message flow](#)
- [Message broker](#)
- [Open source library](#)
- [GOPACS environments](#)
- [Getting started](#)
 - [Account Setup for the GOPACS Application](#)
 - [Set up an API client](#)
 - [Enter your company details for UFTP and OAuth2.0](#)
 - [Obtaining a Bearer token](#)
 - [Implement UFTP endpoint](#)
- [Participant API \('Address book'\)](#)
 - [Search on EAN and UFTP Role](#)
 - [Search on Domain Name and UFTP Role](#)
- [GOPACS on behalf of a Grid Company or Trading Company](#)
- [ISPs](#)
- [Baseline](#)
- [SignedMessage](#)
- [Message types with examples and field specifications](#)
 - [FlexRequest](#)
 - [FlexRequestResponse](#)
 - [FlexOffer](#)
 - [FlexOfferResponse](#)
 - [FlexOrder \(for CLC\)](#)
 - [FlexOrder \(for TDTR\)](#)
 - [FlexOrderResponse](#)
 - [TestMessage & TestMessageResponse](#)
 - [Other message types](#)
- [GOPACS implementation specifics](#)

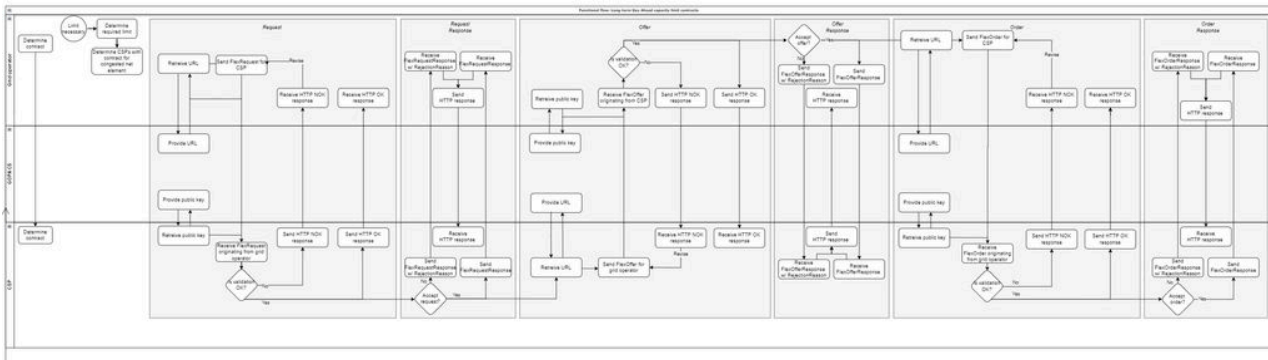
CLC message flow [↗](#)

This requires minimum version 3.0.0, but some improvements are made in 3.1.0. Implementations should adopt version 3.1.0 as soon as possible. [↗](#)

Overview of the GOPACS and UFTP message exchange for CLC contracts. The part where the URL and the public key is retrieved, is GOPACS specific.

Currently, a CLC call (*CBC-afroep*) uses the following message flow:

1. Grid operator calculates prognosis, expects flexibility is needed and initiates a call (*afroep*).
2. `FLexRequest` message is sent by the grid operator to the trading company, requesting a certain amount of flexibility.
3. `FLexRequestResponse` message is sent by the grid operator to the trading company, acknowledging the receipt of the request.
4. Trading company determines what flexibility is available.
5. `FLexOffer` message is sent by the trading company to the grid operator, indicating what flexibility is available.
6. `FLexOfferResponse` message is sent by the grid operator to the trading company, acknowledging the receipt of the offer.
7. Grid operator recalculates and chooses which flexibility to procure.
8. `FLexOrder` message is sent by the grid operator to the trading company, to procure the flexibility.
9. `FLexOrderResponse` message is sent by the trading company to the grid operator, acknowledging the receipt of the order.
10. The call (*afroep*) is now completed.



For a readable version of the above diagram, please check the support documents sections.

TDTR/NFA message flow [↗](#)

This requires version 3.1.0. [↗](#)

Currently, a call (*afroep*) of TDTR/NFA contracts uses the following message flow:

1. Grid operator calculates prognosis, expects flexibility is needed and initiates a call (*afroep*).
2. `FLexOrder` message is sent by the grid operator to the trading company, to procure the flexibility.
3. `FLexOrderResponse` message is sent by the trading company to the grid operator, acknowledging the receipt of the order.
4. The call (*afroep*) is now completed.

There is no negotiation phase where a `FLexRequest` and `FLexOffer` are exchanged.

Message broker [↗](#)

The UFTP protocol allows direct communication between grid operators and trading companies. However, for the sake of monitoring and reporting, UFTP messages should be communicated via the [GOPACS UFTP Message Broker](#). The message broker will forward the messages to the specified UFTP recipient.

Open source library [↗](#)

To aid in building a compliant implementation GOPACS has built an open source library: <https://github.com/shapeshifter/shapeshifter-library>

When implementing the protocol, make sure the implementation is compliant to the GOPACS specifications as described in this document. All UFTP traffic must be uniform in order to make sure that every participant is able to communicate with each other.

GOPACS uses (in this documentation) the term Trading Company for every participant that has a contract with the Grid Company. In the UFTP specification the term AGR (aggregator) is used, and in some Capacity Limit Contracts the term CSP is used.

GOPACS environments [↗](#)

Environment	ACC (suggested for testing integrations)	PRD
GOPACS UI	https://app.acc.gopacs.eu	https://app.gopacs.eu
UFTP endpoint This is where you send your messages to	https://clc-message-broker.acc.gopacs-services.eu/shapeshifter/api/v3/message	https://clc-message-broker.gopacs-services.eu/shapeshifter/api/v3/message
Participant API (address book)	https://clc-message-broker.acc.gopacs-services.eu/v2/participants/	https://clc-message-broker.gopacs-services.eu/v2/participants/
API documentation	Swagger UI	Swagger UI
UFTP Public Key This is the public key that can be used to verify messages sent from GOPACS UI on behalf of a DSO/CSP not having their own implementation.	VFHpQ4B71g0KrVJAG+HK1zQctr1J3zjkk4BYGK79E +C=	pI/s07d9fvoaty05gVLZKLfdqQmdkaMAIgConmcL76U =
Public IPs of GOPACS (for IP whitelisting)	3.75.32.104 3.77.164.86 35.158.231.79	3.121.132.49 3.76.130.111 3.78.82.175
Sign-up link for new CSPs	Sign-up for GOPACS	Sign-up for GOPACS
OAuth2 token endpoint	https://auth.acc.gopacs-services.eu/realms/gopacs/protocol/openid-connect/token	https://auth.gopacs-services.eu/realms/gopacs/protocol/openid-connect/token

Getting started [↗](#)

Account Setup for the GOPACS Application [↗](#)

For New CSPs or Affiliates [↗](#)

If you are a **new** CSP or affiliate (*aangeslotene*), you can create an account yourself if you don't already have one. Please note that each environment requires a **separate account**—accounts are not shared across environments.

After completing the sign-up form and accepting the terms, your request will be sent to a grid operator for approval. Once approved, you will receive an email with instructions to finalize your account setup. Only then will you be able to log in to the GOPACS application.

Set up an API client [↗](#)

After signing up to GOPACS, you need to create at least one **API client** to authenticate your company on API requests:

1. Log in to the GOPACS GUI as company admin
2. Navigate to 'API clients'
3. Create a new API client
4. Store the **client ID** and **client secret** somewhere secure (e.g. secrets manager software). Note that the client secret cannot be viewed again.

Enter your company details for UFTP and OAuth2.0 [↗](#)

Add your company details for UFTP communication and OAuth2.0 on the "My company details" / "Mijn bedrijfsgegevens" page, on the "UFTP" tab.

UFTP details [↗](#)

- UFTP domain (mandatory)
- UFTP endpoint (mandatory, must be https)
- UFTP public key (mandatory)

These details will be your "Participant information" that other companies will retrieve from the GOPACS Participant endpoint to send UFTP requests to your company (see next paragraph "Participant API")

OAuth2.0 [↗](#)

UFTP requests to the GOPACS endpoint must include an [OAuth2 Bearer token](#) in the [Authorization](#) header of each HTTP request. See next paragraph on "Obtaining a Bearer token"

For receiving UFTP requests, using OAuth2.0 is optional. After selecting the optional checkbox to use OAuth2.0 for receiving UFTP requests, the following fields appear:

- Token URL (mandatory)
- Scope (optional, space separated list - scopes that your own authentication provider requires)
- Client Id (mandatory - from set up API client in the previous paragraph)
- Client secret (mandatory - from set up API client in the previous paragraph)

Obtaining a Bearer token [↗](#)

Use the [OAuth2 client credentials flow](#) to obtain a Bearer token:

1. Configure an OAuth2 client to use the GOPACS OAuth2 token endpoint (see the table above)
2. Configure the client ID and client secret of the API client created via the GOPACS GUI
3. Perform the **OAuth2 client credentials** flow
4. A Bearer token will be returned (access token) which will be valid for a limited time (usually 5 minutes)
5. Include the Bearer token in each HTTP request to the GOPACS API
6. Refresh the Bearer token in time by performing the same flow again

Implement UFTP endpoint [↗](#)

As a Trading Company you need to have an implementation of the UFTP protocol for sending and receiving messages to and from grid operators. To receive UFTP messages you need to expose an endpoint compliant to the Shapeshifter UFTP specification, that is ready to accept UFTP messages from GOPACS.

i NB. It is also possible to make use of the GOPACS UI to send and receive UFTP Flex messages.

Participant API ('Address book') [↗](#)

DNS discovery as described in the UFTP specification is currently not supported by GOPACS.

GOPACS provides an alternative API for discovery of UFTP participant information: the Participant API.

There are currently two endpoints for the discovery of UFTP Participant information; 1) Search on EAN and UFTP Role and 2) Search on Domain Name and UFTP Role

Search on EAN and UFTP Role [↗](#)

You can retrieve the participant information by searching on the EAN of the contracted grid connection and the UFTP role of the participant (DSO or AGR). As there may be multiple trading companies/aggregators on one grid connection, the response is a list of participants.

Example Participant API request:

```
1 GET /v2/participants/DSO?contractedEan=123456789012345678 HTTP/1.1
2 Accept: application/json
```

Example Participant API response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
4 [
5   {
6     "domain": "example.com",
7     "publicKey": "VFHpQ4B71g0KrVJAG+HK1zQctr1J3zjkK4BYGK79E+c=",
8     "endpoint": "https://clc-message-broker.gopacs-services.eu/shapeshifter/api/v3/message"
9   }
10 ]
```

NB! Since we actively encourage to let all UFTP message go through the CLC Message Broker for reporting purposes, the returned participant 'endpoint' is always the CLC Message Broker endpoint.

Response Status

HTTP status	Possible cause
200	ok (note: if no participants could be found, an empty list is returned with HTTP Status 200)
400	EAN is not 18 characters wide, or <code>uftpRole</code> is not AGR or DSO (CR0 is not supported)

Search on Domain Name and UFTP Role [↗](#)

This is typically used to verify and decrypt the payload when receiving a message.

i GOPACS also provides a `v1/participants` endpoint, but this API endpoint has been marked as deprecated, and will be removed in a future release; we *strongly* advise you stop using this endpoint, and instead use the v2 endpoint instead.

Example Participant API request:

```
1 GET /v2/participants/DS0/example.com HTTP/1.1
2 Accept: application/json
```

Example Participant API response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
4 {
5   "domain": "example.com",
6   "publicKey": "VFHpQ4B71g0KrVJAG+HK1zQctr1J3zjkk4BYGK79E+c=",
7   "endpoint": "https://uftp.example.com/shapeshifter/v3/message"
8 }
```

Response statuses [↗](#)

HTTP status	Possible cause
200	ok
400	uftpRole is not AGR or DS0 (CR0 is not supported)
404	Participant not found for role + domain name.

GOPACS on behalf of a Grid Company or Trading Company [↗](#)

If a Grid Company or Trading Company doesn't have their own UFTP API, they can use the GOPACS platform (<https://gopacs.eu/>) to send and receive messages on their behalf. In this case, the UFTP messages are delivered to a GOPACS endpoint and GOPACS responds on behalf of that participant.

If GOPACS sends a message on behalf of a participant, the SenderDomain is always the UFTP domain of the actual (original) Grid Company or Trading Company, not GOPACS.

ISPs [↗](#)

Certain message types such as FlexRequest, FlexOffer and FlexOrder contain ISPs: Imbalance Settlement Periods.

While the Shapeshifter UFTP specification allows to include not only the ISPs that are requested (limited or steered), but also those that are available (not limited or steered), GOPACS currently only expects the ISPs that are to be limited under the contract.

ISP Duration [↗](#)

Currently the ISP duration supported is always 15 minutes.

ISP-Duration	Required. Must be PT15M
--------------	-------------------------

ISP Start [↗](#)

First ISP of the day is 1. 00:00:00 (inclusive) until 00:15:00 (exclusive)

Second ISP of the day is 2. 00:15:00 (inclusive) until 00:30:00 (exclusive)


Last ISP of the day is 96 or 100 or 92. 23:45:00 (inclusive) until 00:00:00 the next day (exclusive)

ISPs and Daylight Saving time [↗](#)

ISPs with respect to Daylight Saving Time (DST), assuming Europe/Amsterdam and a 15 minute ISP duration:

- On the last Sunday of March when the clock goes from CET (standard) to CEST (summer), the number of ISPs will be 92:
ISP 1: 00:00-00:15
ISP 2: 00:15-00:30
ISP 3: 00:30-00:45
ISP 4: 00:45-01:00
ISP 5: 01:00-01:15
ISP 6: 01:15-01:30
ISP 7: 01:30-01:45
ISP 8: 01:45-**03:00**
ISP 9: **03:00**-03:15
etc.
ISP 92: 23:45-00:00
- On the last Sunday of October when the clock goes from CEST (summer) to CET (standard), the number of ISPs will be 100.
ISP 1: 00:00-00:15
ISP 2: 00:15-00:30
ISP 3: 00:30-00:45
ISP 4: 00:45-01:00
ISP 5: 01:00-01:15
ISP 6: 01:15-01:30
ISP 7: 01:30-01:45
ISP 8: 01:45-02:00
ISP 9: 02:00-02:15
ISP 10: 02:15-02:30
ISP 11: 02:30-02:45
ISP 12: 02:45-**02:00**
ISP 13: **02:00**-02:15
etc.
ISP 100: 23:45-00:00
- On any other day, the number of ISPs will be 96.

ISP (Min, Max) Power [↗](#)

 This is the **absolute** capacity in watts, not the deviation as stated in the Shapeshifter specification. This might change when V4.x of the Shapeshifter specification is implemented.

- The GUI shows the power values in MW
- In the UFTP messages, power values are in W

Depending on the capacity limiting direction:

- When limiting **feed-in**:
 - `MaxPower` = 0
 - `MinPower` is the maximum allowed feed in as a negative number. E.g. when limiting the feed in to 3 MW, `MinPower` = -3000000.
If no feed in is allowed at all, `MinPower` = 0
 - `MinPower` must be a multiple of -1000W (in other words, the limitation is specified in steps of 1kW) and must also match the steps defined in the contract.
- When limiting **offtake**:

- `MinPower` = 0
- `MaxPower` is a positive number specifying the maximum allowed power offtake in Watts.
- `MaxPower` must be a multiple of 1000W.

Baseline [↗](#)

This is a proposal and is not supported yet.

<p>ISP.DefaultBaseline</p> <p>Not supported yet ↗</p>	<p>Capacity in the default situation that would occur if no flexibility were activated. This is usually equal to the contracted transport capacity (GTV).</p> <p>Example: the GTV is 100 MW. No flexibility is activated yet: the <code>DefaultBaseline</code> is 100 MW. If 25 MW of flexibility was already activated: the <code>DefaultBaseline</code> is still 100 MW.</p> <p>This is a proposal and is not supported yet.</p>
<p>ISP.Baseline</p> <p>Not supported yet ↗</p>	<p>Capacity baseline before this flexibility was requested. If flex capacity was activated earlier, in a different conversation, then this is the capacity with the deviation applied.</p> <p>Example: the <code>DefaultBaseline</code> is 100 MW. An earlier conversation has activated 25 MW of flexibility. When this <code>FlexRequest</code> is sent to request the activation of another 20 MW of flexibility, the <code>Baseline</code> would be 75 MW and the <code>MaxPower</code> would be 55 MW.</p> <p>This is a proposal and is not supported yet.</p>

SignedMessage [↗](#)

Example of a SignedMessage HTTP request (some headers omitted for clarity):

```

1 POST /shapeshifter/api/v3/message HTTP/1.1
2 Accept: application/json (or */* or omit)
3 Authorization: Bearer ...
4 Content-Type: text/xml
5
6 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
7 <SignedMessage
8   SenderDomain="dso.nl"
9   SenderRole="DS0"
10  Body="..." />

```

Attribute	GOPACS expectation
SenderDomain	<p>Grid Company must log in to the GOPACS UI and configure UFTP domain, public key and endpoint in GOPACS, prior to receiving or sending messages.</p> <p>When GOPACS is sending messages on behalf of Grid Company or Trading Company, then the SenderDomain is equal to the domain of actual UFTP participant.</p>
SenderRole	<p>Must be <code>DS0</code> for a Grid Company.</p> <p>Must be <code>AGR</code> for a Trading Company.</p> <p>Other values are not supported.</p>

	<p>GOPACS is not considered a 'party' in the UFTP exchange and therefore does not have a 'role'.</p> <p>When GOPACS is sending messages on behalf of Grid Company or Trading Company, then the SenderRole is equal to the role of actual UFTP participant.</p>
Body	<p>Base64 encoded payload that will be decrypted using <code>crypto_sign_open</code> (see Public-key signatures Libsodium documentation). GOPACS does not support sealing and unsealing of messages (just signing).</p> <p>GOPACS uses the SenderDomain and SenderRole to lookup the public key that is used for verifying the signature of the message.</p> <p>Encryption-in-transit is covered by enforcing TLS connections.</p> <p>Encryption-at-rest should be done using common security practices and tools by the implementing system.</p> <p>GOPACS uses Lazysodium which is a Java wrapper over the Libsodium library.</p>

Response statuses [↗](#)

HTTP status	Possible cause
200	Message has correct signature, was XSD valid and will be processed asynchronously.
400	<p>Wrong content type</p> <p>Technical XSD validation error (no functional validations yet)</p> <p>Error during XML deserialization</p>
401	<p>Bearer token not provided or invalid.</p> <p>Public key of sender not found or incorrect.</p> <p>Message signature could not be verified with public key of sender.</p>
403	Bearer token not authorized to perform this request.
5xx	Unexpected (temporary) error on GOPACS side.

Message types with examples and field specifications [↗](#)

FlexRequest [↗](#)

Sent by the Grid Company to the Trading Company.

Example of a FlexRequest message for a CLC contract, where the contracted capacity is 100 MW, the requested flex capacity is 50 MW:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexRequest
3   Version="3.0.0"
4   SenderDomain="dso.nl"
5   RecipientDomain="agr.nl"
6   TimeStamp="2021-10-29T06:54:26.861Z"
7   MessageID="d3ae4836-55b1-4084-b54e-34107b22648c"
8   ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9   ServiceType="CBC"
10  ISP-Duration="PT15M"
11  TimeZone="Europe/Amsterdam"
12  Period="2021-10-30"
13  ContractID="A-AA-A-12345"
14  CongestionPoint="ean.265987182507322951"
15  Revision="1"

```


ServiceType	The ServiceType specifies which type of flexibility is being requested. <table border="1" data-bbox="423 163 1438 281"> <tr> <td>ServiceType</td> <td></td> </tr> <tr> <td>CBC</td> <td>Capacity limiting contract (<i>capaciteitsbeperkingscontract</i>)</td> </tr> </table>	ServiceType		CBC	Capacity limiting contract (<i>capaciteitsbeperkingscontract</i>)
ServiceType					
CBC	Capacity limiting contract (<i>capaciteitsbeperkingscontract</i>)				
ISP-Duration	Required. See ISPs .				
TimeZone	Required. Must be Europe/Amsterdam				
Period	Required. ⚠ The day of congestion. Format: YYYY-MM-DD always interpreted in Europe/Amsterdam time zone. <u>Any offset is ignored</u> . The message must be sent before 12:00:00 the day before Period. Examples: <ul style="list-style-type: none"> • If the message is received before 12:00:00, then the Period may be tomorrow or later. • If the message is received after 12:00:00, then the Period must be the day after tomorrow or later. 				
ExpirationDateTime	⚠ The expiration date time must be no later than 12:00:00 the day before the day of congestion (Period).				
ContractID	Functional required. Typical format: A-AA-A-12345 Trading Company must log in to the GOPACS UI and register the CLC contract, prior to receiving or sending messages.				
CongestionPoint	ean. [0-9]{18} Must be a known EAN of a preregistered CLC contract in GOPACS. Does <u>not</u> have to be known as Grid Connection in GOPACS.				
ServiceType	Optional. Ignored.				
ISP	Required. See ISPs .				
ISP.Start	Required. See ISPs .				
ISP.Duration					
ISP.Disposition	Must be Requested See ISPs .				
ISP.MinPower	See ISPs .				
ISP.MaxPower					

FlexRequestResponse [🔗](#)

Example FlexRequestResponse:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexRequestResponse
3   Version="3.0.0"
4   SenderDomain="agr.nl"
5   RecipientDomain="dso.nl"
6   TimeStamp="2021-10-29T06:54:36.4437962Z"
7   MessageID="7f0f4e68-f842-4b92-911e-b26f85525067"
8   ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9   Result="Accepted"
10  FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c"/>

```

Example of the same FlexRequestResponse message signed with the ACC key:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <SignedMessage SenderDomain="agr.nl" SenderRole="AGR"
3     Body="3fD4Ie5Jk6h7k6TQaEsJ4Vego1CXpA/1ztXlyaej1db0SnuFscFZZy630EsUJFVM3Ihy0+3
4     V9WdcW0R5LdnJATxGbGV4UmVxdWVzdFJlc3BvbnNlDQogICAgICAgIFZlcnNpb249IjMuMC4wIgw0K
5     ICAgICAgICBTZW5kZXJEb21haW49ImFnci5ubCINCiAgICAgICAgUmVjaXBpZW50RG9tYwluPSJkc
6     28ubmwiDQogICAgICAgIFRpbWVtdGFtcD0iMjAyMS0xMC0yOVQwNjoiND0zNi40NDM3OTYywiINCi
7     AgICAgICAgTWFzc2FnZULEPSI3ZjBmNGU2OC1mODQyLTRiOTItOTExZS1iMjZmODU1MjUwNjciDQo
8     gICAgICAgIENvbnZlcnNhdGlvbklEPSi00GNkYzNkMi01NmMwLTQzNmMt0GQ1YS02ZjZjYzNkYzUz
9     OGQiDQogICAgICAgIFJlc3VsdD0iQWVjZXB0ZWQidQogICAgICAgIEZsZXhSZXF1ZXN0TWFzc2FnZ
10    ULEPSJkM2FLNDgzNi01NWlxLTQwODQtYjU0ZS0zNDEwN2IyMjY0OGMiLz4=" />

```

Or when it is rejected:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexRequestResponse
3     Version="3.0.0"
4     SenderDomain="aggregator.org"
5     RecipientDomain="uftp.dso.nl"
6     TimeStamp="2021-10-29T06:54:36.4437962Z"
7     MessageID="7f0f4e68-f842-4b92-911e-b26f85525067"
8     ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9     Result="Rejected"
10    RejectionReason="Reference Period mismatch"
11    FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c" />

```

Example of the same FlexRequestResponse message signed with the ACC key:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <SignedMessage SenderDomain="aggregator.org" SenderRole="AGR"
3     Body="+hM5FFli0QEpuMLwVpy6KKhb80x0trCu//VhLYnhiJAwZC8Eh743SPERlsPR9bpWTJdnuSfd
4     SY3UamQuafGVCzxGbGV4UmVxdWVzdFJlc3BvbnNlDQogICAgICAgIFZlcnNpb249IjMuMC4wIgw0KIC
5     AgICAgICBTZW5kZXJEb21haW49ImFnZ3JlZ2F0b3Iub3JnIgw0KICAgICAgICBSZWNPcGllbnREb21h
6     aW49InVmdHAuZHNvLm5sIgw0KICAgICAgICBUaW1lU3RhbXA9IjIwMjE0MDY2MDY2MDY2MDY2MDY2
7     QzNzk2MloiDQogICAgICAgIE1lc3NhZ2VJRd0iN2YwZjRlNjgtZjg0Mi00YjkyLTkxMwUtYjI2Zjg1
8     NTIiMDY3Igw0KICAgICAgICBDb252ZXJzYXRpb25JRd0iNDhjZGMzZDIiNTZjMjMzZjMzZjMzZjMz
9     Y2Y2MzZGM1MzhkIgw0KICAgICAgICBSZXN1bHQ9IjIwMjE0MDY2MDY2MDY2MDY2MDY2MDY2MDY2
10    ZWFzb249IjIwMjE0MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2MDY2
11    FnZULEPSJkM2FLNDgzNi01NWlxLTQwODQtYjU0ZS0zNDEwN2IyMjY0OGMiLz4=" />

```

FlexOffer [🔗](#)

Sent by the Trading Company to the Grid Company as answer to a FlexRequest.

Example of a FlexOffer message for a CBC contract:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOffer
3     Version="3.0.0"
4     SenderDomain="agr.nl"
5     RecipientDomain="dso.nl"
6     TimeStamp="2021-10-29T06:54:36.8868538Z"
7     MessageID="338ed243-5517-4400-962e-2b7b812c468c"
8     ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9     ServiceType="CBC"
10    ISP-Duration="PT15M"
11    TimeZone="Europe/Amsterdam"
12    Period="2021-10-30"

```

```

13 CongestionPoint="ean.265987182507322951"
14 ExpirationDateTime="2021-10-29T10:30:00Z"
15 FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c"
16 ContractID="A-AA-A-12345"
17 BaselineReference=""
18 Currency="EUR">
19 <OfferOption OptionReference="ba40a5f8-849b-4fe6-958f-e628a1653558"
20   Price="0.00">
21   <ISP Start="58" Duration="1" Power="50000000"/>
22   <ISP Start="59" Duration="1" Power="50000000"/>
23   <ISP Start="60" Duration="1" Power="50000000"/>
24   <ISP Start="61" Duration="1" Power="50000000"/>
25 </OfferOption>
26 </FlexOffer>

```

Example of the same FlexOffer message signed with the ACC key:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <SignedMessage SenderDomain="agr.nl" SenderRole="AGR"
3   Body="whfSzCmP90ml2y/6lfz/7a6hRvz7Am5waAwmtKT52kbEG90hqHTj4DisCRNmGHrmdjAsq6AyNB/YG+gDnZ5BjxGbGV4

```

⚠ Unsolicited FlexOffer messages are rejected by GOPACS. There must always be a preceding FlexRequest.

⚠ At most 1 FlexOffer message may be sent as part of a conversation. All successive FlexOffer messages will be rejected by GOPACS.

Attribute/Element	
D-PrognosisMessage ID	Ignored.
BaselineReference	Ignored.
Currency	Must be EUR .
Price	Must be 0.00 (for now). Must comply with ISO 4217 . 0.00 , 0.0 and 0 are all allowed and considered equal in the GOPACS implementation.
OfferOption	⚠ Exactly 1 OfferOption element is expected.
OfferOption.MinActivationFactor	Optional. Ignored.
ISP.Power	Depending on the contractual agreements between AGR and DSO the Power either is equal to what has been requested in the FlexRequest or it can deviate. In the case of a limitation on production, Power refers to the MinPower attribute of the ISP . In case the consumption is limited, Power refers to the MaxPower attribute. See ISPs . It is allowed to send an offer on a subset of the requested ISPs.

Other attributes like Period, CongestionPoint, ContractID, etc. must be equal to the original FlexRequest.

FlexOfferResponse [🔗](#)

Example FlexOfferResponse:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>

```

```

2 <FlexOfferResponse
3   Version="3.0.0"
4   SenderDomain="dso.nl"
5   RecipientDomain="arg.nl"
6   TimeStamp="2021-10-29T06:54:36.4437962Z"
7   MessageID      = UUID
8   ConversationID = UUID
9   FlexOfferMessageID = UUID
10  Result          = ("Accepted" | "Rejected")
11  RejectionReason = String (Only if Result = "Rejected")
12 />

```


FlexOrder (for CLC) [↗](#)

Example of a FlexOrder message for a CLC contract, where the off-take transport capacity is limited to 50 MW (the original contracted transport capacity is 100 MW):

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOrder
3   Version="3.0.0"
4   SenderDomain="dso.nl"
5   RecipientDomain="agr.nl"
6   TimeStamp="2021-10-29T06:55:36.518Z"
7   MessageID="dc0f19c4-3835-4753-8f0c-0319d6642fbb"
8   ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9   ServiceType="CBC"
10  ISP-Duration="PT15M"
11  TimeZone="Europe/Amsterdam"
12  Period="2021-10-30"
13  CongestionPoint="ean.265987182507322951"
14  FlexOfferMessageID="338ed243-5517-4400-962e-2b7b812c468c"
15  ContractID="A-AA-A-12345"
16  Price="0.00"
17  Currency="EUR"
18  OrderReference="None">
19  <ISP Start="58" Duration="1" Power="50000000"/>
20  <ISP Start="59" Duration="1" Power="50000000" />
21  <ISP Start="60" Duration="1" Power="50000000"/>
22  <ISP Start="61" Duration="1" Power="50000000"/>
23 </FlexOrder>

```

Attribute/Element	
ServiceType	CBC
Currency	Must be <code>EUR</code> .
Price	Must equal the Price in the FlexOffer.
OrderReference	May be filled by the calling grid company for settlement process. If the grid operator is using GOPACS for UFTP, and they start a request from the GUI, this field is filled with a generated UUID.
ISP	 Currently GOPACS orders exactly what was offered (if on behalf of a Grid Company) - including ISPs and min activation factor.
ISP.Power	Depending on the contractual agreements between AGR and DSO the <code>Power</code> either is equal to what has been requested in the <code>FlexRequest</code> or it can deviate.

In the case of a limitation on production, power refers to the `MinPower` attribute of the `FlexRequest`. In case the consumption is limited, power refers to the `MaxPower` attribute. See [ISPs](#).

FlexOrder (for TDTR) [↗](#)

This requires version 3.1.0. [↗](#)

Example of a FlexOrder message for a TDTR contract, where the transport capacity is limited to 50 MW (the original contracted transport capacity is 70 MW):

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOrder
3   Version="3.1.0"
4   SenderDomain="dso.nl"
5   RecipientDomain="agr.nl"
6   TimeStamp="2021-10-29T06:55:36.518Z"
7   MessageID="dc0f19c4-3835-4753-8f0c-0319d6642fbb"
8   ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9   ServiceType="TDTR"
10  ISP-Duration="PT15M"
11  TimeZone="Europe/Amsterdam"
12  Period="2021-10-30"
13  CongestionPoint="ean.265987182507322951"
14  ContractID="0000001"
15  Price="0.00"
16  Currency="EUR"
17  OrderReference="None">
18  <ISP Start="58" Duration="1" Power="50000000"/>
19  <ISP Start="59" Duration="1" Power="50000000"/>
20  <ISP Start="60" Duration="1" Power="50000000"/>
21  <ISP Start="61" Duration="1" Power="50000000"/>
22 </FlexOrder>

```

Attribute/Element							
Version	Must be 3.1.0 for ATR.						
ServiceType	Type of ATR contract. <table border="1" data-bbox="418 1388 1438 1562"> <thead> <tr> <th colspan="2">ServiceType</th> </tr> </thead> <tbody> <tr> <td>TDTR</td> <td>Time-bounded transport right (<i>tijdsduurbonden transportrecht</i>)</td> </tr> <tr> <td>NFA</td> <td>Non firm transport right (<i>non-firm ATO</i>)</td> </tr> </tbody> </table>	ServiceType		TDTR	Time-bounded transport right (<i>tijdsduurbonden transportrecht</i>)	NFA	Non firm transport right (<i>non-firm ATO</i>)
ServiceType							
TDTR	Time-bounded transport right (<i>tijdsduurbonden transportrecht</i>)						
NFA	Non firm transport right (<i>non-firm ATO</i>)						
ContractID	Unique number of the ATR contract.						
Currency	Must be EUR.						
Price	Must be 0.00. But is ignored for now.						
OrderReference	May be filled by the calling grid company for settlement process.						
ISP	ISPs that are to be limited under the contract. See ISPs of <code>FlexRequest</code> .						
ISP.Power	See ISPs .						

FlexOrderResponse [↗](#)

i An "Accepted" response from the Trading Company means that there is a binding agreement.

Example FlexOrderResponse:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOrderResponse
3   Version           = "3.0.0"
4   SenderDomain      = "dso.nl"
5   RecipientDomain   = "arg.nl"
6   TimeStamp         = "2021-10-29T06:54:36.4437962Z"
7   MessageID         = UUID
8   ConversationID    = UUID
9   FlexOrderMessageID = UUID
10  Result             = ("Accepted" | "Rejected")
11  RejectionReason    = String (Only if Result = "Rejected")
12 />
```

TestMessage & TestMessageResponse [↗](#)

The UFTP protocol provides the `TestMessage` and `TestMessageResponse` message types for testing purposes. Both are Supported by the `clc-message-broker`. They can be used to test your UFTP settings and to test sending an receiving messages and message responses.

Currently, this is only supported for API, not on the GUI, it is not possible to trigger a test message on the GOPACS GUI.

Testing sending messages and receiving a response message [↗](#)

If a `TestMessage` is sent to the `clc-message-broker` with a recipient that is using GOPACS for UFTP, a `TestMessageResponse` will be sent back automatically.

Test receiving messages and responding with a response message [↗](#)

To test receiving a `TestMessage` and responding with a `TestMessageResponse` you can ask your grid company to send a `TestMessage` with your uftp implementation as recipient. The `clc-message-broker` will then forward this test message to your uftp implementation.

Other message types [↗](#)

- **✘** FlexOfferRevocation not supported yet
- **✘** FlexRequest revisions not supported yet
- **✘** Other message types are not supported yet

GOPACS implementation specifics [↗](#)

⚠ A duplicate MessageID is immediately responded to with a 400 Bad Request and **not** a 200 OK followed by "Rejected" response as described in the specification!

⚠ MinPower, MaxPower and Power are implemented as absolute values.

- After a 200 OK is returned, a received message is immediately processed by GOPACS. An accepted or rejected response is sent back almost instantaneously.
- The user receives realtime email notifications when a FlexRequest, FlexOffer or FlexOrder is received, rejected or failed to deliver.
- An outgoing UFTP message is retried every 3 minutes for a maximum of 5 tries. After that, the user and GOPACS DevOps team are notified of a failure to deliver. Specifically on a 400 Bad Request, a message is not retried.
- Typically there will be at most 15 mins between FlexRequest and FlexOrder.